

Système de cache HTTP distribué pour réseaux informatiques maillés, instables, et à forts temps de latence.

(Pré-étude)

Christopher Glass

Etudiant en emploi, travaillant chez VMware Switzerland Sàrl.

Mandant:

Ingénieurs du Monde, Sous la direction du Dr. François Ullmann

Conseiller:

Dr. Eytan Zysman

5 juin 2009

Table des matières

1	Introduction	4
1.1	Contexte	4
1.1.1	Ingénieurs du Monde	4
1.1.2	Réseaux hertziens	4
1.1.3	Coût peu élevé des antennes directionnelles	5
1.1.4	Grande disponibilité des antennes GSM	5
1.2	Solutions alternatives	6
1.2.1	Squid	6
1.2.2	Dijjer	7
1.2.3	Cache IP	7
2	Objectifs	8
2.1	Objectif principal	8
2.2	Objectifs secondaires	8
2.3	Facteurs de risque	8
2.3.1	Risque principal	8
2.3.2	Risque secondaire	9
3	Cahier des charges détaillé	10
3.1	Livrables	10
3.1.1	Logiciel	10
3.1.2	Documentation	10
3.2	Fonctions souhaitées	10
3.3	Contraintes	11
4	Environnement de travail prévu	12
4.1	Matériel et outils	12
4.2	Logiciel	12
4.2.1	Langage de programmation	12
4.2.2	Système d'exploitation	13
4.2.3	Base de données	13
4.3	Banc de test	13
4.3.1	Hôtes physiques	14
4.3.2	Hôte unique, multiples instances	14
4.3.3	Hôtes virtuels	14
4.3.4	Solution hybride	14
4.3.5	Solution retenue	14
5	Planification	15
5.1	Plan de travail détaillé	15
5.1.1	Cas d'utilisation	15
5.1.2	Découpage du travail	15
5.1.3	User stories	16
5.1.4	Notes sur les user stories	17
5.1.5	Découpage temporel	18

5.1.6	Découpage de la phase de réalisation	18
5.1.7	Assignation des user stories aux itérations	18
5.2	Échéancier	20
6	Conclusion	21
6.1	Remerciements	21
6.2	Pourquoi ce projet ?	21
7	Exploitation de sources bibliographiques	23
7.1	Webographie	23
7.2	Bibliographie	23
8	Glossaire	24

Chapitre 1

Introduction

1.1 Contexte

Dans les pays émergents, un des principaux obstacles au déploiement de l'internet est le coût prohibitif des infrastructures fondamentales de liens point-à-point¹ entre les différentes parties du pays.

En effet, déployer un réseau de type occidental requiert d'installer des câbles de fibre optique sur de longues distances, qui le plus souvent sont faiblement peuplées, et qui potentiellement traversent des terrains inhospitaliers (chaînes montagneuses, déserts ...).

Partant de ce constat, j'ai entrepris d'évaluer une solution logicielle pour améliorer la qualité du réseau internet dans des régions faiblement peuplées. Soumis à Ingénieurs du Monde, le projet a suscité un grand enthousiasme, ce qui m'a encouragé à le réaliser dans le cadre de mon travail de diplôme.

1.1.1 Ingénieurs du Monde

Ingénieurs du Monde (IdM), est une organisation non-gouvernementale enregistrée et représentée auprès de l'organisation des nations unies dont la vocation est d'améliorer le bien-être et le développement des pays émergents et des régions en voie de développement².

Notamment présent en Afrique, IdM sponsorise plusieurs écoles techniques, notamment en Tanzanie, connectées à l'internet ou pas, qui bénéficieront directement de la mise en place d'un réseau de caches internet³ autant par l'utilisation de celui-ci que par l'acquisition de connaissances apportées sous forme de documentation.

1.1.2 L'avantage des réseaux hertziens

Bien que déployer de l'infrastructure câblée soit généralement trop coûteux, le déploiement d'antennes-relais pour une infrastructure internet est comparativement abordable.

Les antennes, particulièrement en terrain inhospitalier, posent cependant problème : leur taux moyen de panne est bien plus élevé, de par la nature des antennes exposées et de leur sensibilité face aux éléments (pluie, neige, tempêtes de sable, chaleur excessive, nuages de sauterelles ...).

Dans ce contexte, un système distribué habituel ne fonctionne que très mal : les caches habituels partent du principe que bien que la bande passante⁴ soit limitée, les temps de latence⁵ sont bas et les noeuds généralement disponibles. Pour un déploiement dans un milieu

¹Les liens points-à-points relient entre eux deux interlocuteurs uniques

²Vous pouvez trouver de plus amples informations sur leur site internet cité en webographie (Section 7.1)

³Cache est ici utilisé dans son sens "réserve temporaire" - Voir glossaire (8)

⁴Bande passante : Quantité d'information transmissible dans un temps donné. Voir glossaire (8)

⁵Temps que met l'information à parcourir un aller-retour entre deux interlocuteurs - Voir glossaire (8)

inhospitalier, la probabilité que la bande-passante soit le goulot d'étranglement diminue considérablement.

Réseaux point-à-points Le logiciel de cache distribué présenté amoindrit ce problème :

- L'information est distribuée le long des noeuds
- Il est possible de paramétrer le niveau de récursion⁶.

Un unique lien composé de plusieurs caches "en série" distribue l'information au sein de chaque noeud, et ainsi tire profit de la somme des ressources *hardware* qui le compose.

Réseaux maillés Le réseau maillé est plus intéressant, car chaque noeud possède plusieurs liens vers des noeuds alentours.

Chacun des noeuds profite ainsi de la "proximité" relative de ses voisins pour aller y chercher, à moindre coût, les informations demandées par ses utilisateurs. Partant du postulat que les sites auxquels on accède sont globalement les mêmes (Wikipedia ou Google par exemple, seront vraisemblablement consultés dans plusieurs écoles voisines avec une haute fréquence), la proximité des noeuds, au sens "nombre de liens à parcourir", collabore à la qualité du service internet proposé aux utilisateurs.

1.1.3 Coût peu élevé des antennes directionnelles

Les antennes directionnelles sont vastement disponibles, à faible coût. En effet, les antennes réceptrices de signaux satellites sont peu coûteuses, et du mode de vie occidental découle qu'une grande quantité de ces antennes finissent chaque année dans des centres de recyclage, au mieux, ou dans des décharges, au pire.

Se servir de ces antennes de récupération permet donc de fabriquer des antennes WiFi directionnelles efficaces et à coût raisonnable. Le site www.weiand.nl propose par exemple une solution dont le coût est inférieur à 10 Euros (Voir webographie en Section 7.1).

1.1.4 Grande disponibilité d'antennes GSM

Dans le monde, les réseaux téléphoniques sont fortement disséminés. Même dans les pays en voie de développement, bien que le niveau de vie moyen soit largement moins élevé que dans les pays occidentaux, les téléphones cellulaires sont très répandus.

Il est donc logique, dans une optique de distribution de connections internet, de tirer parti des infrastructures existantes afin d'amoindrir les coûts au maximum.

Il est tout à fait possible, voir facile, de coupler un module GPRS à un ordinateur à portée de réception d'une antenne téléphonique, et de créer à partir de celui-ci un réseau WiFi, directionnel ou pas, afin de distribuer plus avant la connection internet ainsi rendue disponible.

Dans la mesure où le système de cache distribué décrit dans ce texte est indifférent au mode de transmission ou de routage, il est trivial d'en installer une copie sur l'ordinateur, devenu "gateway"⁷ internet, et de configurer les caches des noeuds voisins pour le considérer comme un cache pair. Il peut, de plus, fournir un point de routage vers le réseau internet externe. Un réseau maillé configuré avec un protocole intelligent (de type OSPF⁸ par exemple), peut tirer parti d'un tel système de routage.

⁶Profondeur d'exploration de l'arbre recouvrant ayant N pour sommet. Voir glossaire (8)

⁷Un noeud d'un réseau équipé pour s'interfacer avec un autre réseau, éventuellement en utilisant un protocole différent. Voir glossaire chapitre 8

⁸*Open Shortest Path First* - Voir glossaire chapitre 8

1.2 Solutions alternatives

Une question valable que l'on peut se poser en lisant ce document est “pourquoi ne pas simplement utiliser le logiciel de cache X, qui existe déjà ?”. Il existe plusieurs raisons à cela :

1.2.1 Squid

Le plus célèbre des logiciels de cache.

Squid est un logiciel *libre* de cache HTTP très complet, permettant de répondre efficacement à beaucoup de scénarios d'utilisation. Plus d'informations sur Squid peut être obtenue en consultant leur site web cité en Webographie (Section 7.1).

Avantages

- Cache de contenu pour une utilisation locale. Squid permet de *mettre en cache* des objets HTTP pour une utilisation locale, c'est à dire dans le même type de configuration que le logiciel présenté, ou un grand nombre d'utilisateurs accèdent à des sites similaires, et où Squid sert le contenu le plus statique (images, scripts, pages html) depuis son *cache* local plutôt que d'aller chercher l'information sur le site distant à chaque fois.
- Cache de contenu local pour une utilisation distante. Squid permet de mettre en cache des données d'un serveur web local sur d'autres machines locales exposées à l'internet, afin de répartir la charge web. Cette configuration permet de ne pas surcharger le serveur web local tout en garantissant aux utilisateurs externes des temps de réponse raisonnables. En effet, les serveurs Squid absorbent une majorité des demandes, et n'envoient au “vrai” serveur web que les requêtes qu'ils n'arrivent pas à traiter. C'est le mode d'utilisation que les grands sites font de Squid (par exemple Wikipedia, Amazon.com).

Inconvénients

- Structure hiérarchique. Bien que Squid puisse fonctionner dans une configuration “réseau de caches”, il n'est capable que d'établir des relations hiérarchiques entre caches, non des relations de type “pairs”. Ainsi, dans un réseau maillé, il serait indispensable de segmenter le réseau en plusieurs sous-réseaux de caches indépendants, et surtout, il faudrait définir un cache “père” auquel tout les caches “fils” vont demander l'information quand ils ne l'ont pas ou qu'elle n'est pas considérée assez “fraîche”⁹. Dans un réseau instable, il est fort dangereux de créer des points de pannes critiques - si un seul “père” tombe en panne, tout le sous-réseau de cache est alors inutilisable.
- Protocole spécialisé. Squid utilise un protocole spécialisé pour synchroniser les caches entre eux. Cela a bien sûr l'avantage de garantir des données plus fraîches, mais aussi l'échange de statistiques sur les objets les plus demandés afin de les répartir de façon plus efficace. En revanche, Squid part du principe que les temps de latence sont très faibles, et réduit un maximum l'utilisation de la bande-passante. C'est tout à fait logique : Squid est conçu pour des réseaux occidentaux, où la latence est minime (les réseaux sont très souvent en fibre optiques), et dans le cas d'un cache de contenu pour utilisateurs distants, il est normal de vouloir minimiser la bande-passante utilisée - c'est justement pour cela que Squid a été déployé !
- Difficulté d'apprentissage. Bien que le code source de Squid soit disponible, sa taille, ainsi que le langage de programmation utilisé rendent l'apprentissage de son fonctionnement interne difficile. La configuration d'un cache Squid est elle-même peu aisée, particulièrement dans le cas de réseaux de caches.

⁹La fraîcheur dans ce contexte, désigne l'actualité d'un fichier. Un fichier “frais” est un fichier dont on considère le contenu comme actuel.

1.2.2 Dijjer

Dijjer utilise le protocole bittorrent¹⁰ de façon très astucieuse en l'appliquant aux caches Web. Presque entièrement décentralisé, Dijjer propose à ses utilisateurs de mettre en cache sur leurs machines le contenu internet qu'ils obtiennent durant la navigation Internet¹¹, puis de mettre les utilisateurs en relation à travers le protocole bittorrent, avec lequel ils mettent en commun les données qu'ils ont accumulés.

Plus d'information sur Dijjer peut être obtenue en consultant le site web cité en Webographie (Section 7.1).

Avantages

- Sauvegarde de la bande-passante. Dijjer est a été imaginé pour conserver au maximum la bande passante des utilisateurs.
- Mise en relation automatique des caches. Les caches Dijjer utilisent le protocole Bittorrent : une machine centrale met les caches en relation, afin de créer un énorme réseau d'utilisateurs qui partagent mutuellement leurs données *via* bittorrent. Si du contenu est disponible sur d'autre caches, Dijjer va alors télécharger le contenu, morceau par morceau, depuis la machine de *plusieurs utilisateurs à la fois*. Le gain de temps est énorme, et de plus, le fait que le site distant "original" soit disponible ou pas n'a aucune importance (il n'est parfois pas sollicité du tout!).
- Contenu distribué. Chaque machine n'utilise pas plus d'espace disque qu'habituellement - les navigateurs modernes mettent déjà automatiquement en cache le contenu affiché. Dijjer se contente de mettre en commun ce cache.

Inconvénients

- Centralisé. Bien que faiblement centralisé, Dijjer dépend d'un *tracker* central. Ne pas pouvoir atteindre ce serveur central implique une connectivité avec les autres caches fortement réduite, voir impossible.
- Fort nombre de requêtes. Le protocole BitTorrent a été imaginé pour distribuer de gros fichiers de façon efficace, sur des réseaux de type occidentaux (au pire, composés de connections de type DSL). Les temps de latence étant en général négligeables, Dijjer est très bavard, même si la quantité d'information échangée à chaque fois est négligeable. Dans un réseau à forte latence, ce bavardage incessant avec les autres caches entraînerait un fort ralentissement du réseau pour tous les utilisateurs, et conduirait à un taux de pannes généralement plus élevé.
- Plus maintenu. Dijjer est pour l'instant "orphelin" dans la mesure où son développeur principal n'a apparemment plus le temps de le maintenir (en date du 1er Juin 2009). L'évolution et la survie du projet n'est donc pas garantie.

1.2.3 Caches IP

Sous cette appellation générique, nous considérerons les caches de couche ISO 3 - les caches fonctionnant au niveau de la trame TCP/IP. Ces caches fonctionnent en cachant les transactions TCP entre deux ordinateurs. Bien que compatibles avec la solutions proposée, les caches TCP sont efficaces pour un lien unique, mais ne sont pas distribués.

En effet, il est illogique de distribuer du cache de trames TCP/IP, dans la mesure où les demandes (*lookups*) cache et leurs réponses sont elle-mêmes des trames TCP/IP ! Une simple demande, fructueuse ou pas, entraînerait un nombre immense d'envoi de trames TCP/IP, qui elles-mêmes feraient l'objet de demandes de cache, qui à leur tour entraîneraient un nouvel envoi massif de requêtes, ainsi de suite ...

¹⁰Le protocole bittorrent est généralement utilisé pour distribuer de gros fichiers à un grand nombre de gens. Voir Glossaire, section 8

¹¹Les navigateurs Web mettent en cache une grande quantité de données automatiquement.

Chapitre 2

Objectifs

2.1 Objectif principal

L'objectif principal de ce travail est de fournir une “brique de base” pour permettre un déploiement à grande échelle, dans un pays en voie de développement, d'un réseau internet à faible coût : un logiciel de cache HTTP distribué pour réseaux instables et à forts temps de latence.

Le cache proposé aura en outre les qualités suivantes :

Persistance Il est nécessaire, étant donnée la fréquence des pannes dans l'environnement de déploiement envisagé, qu'une remise en service d'un noeud de le laisse pas “vide” de contenu, afin qu'il puisse immédiatement contribuer aux caches voisins.

Efficacité Le logiciel proposé devra être peu demandant en ressources physiques, telles que temps processeur ou quantité de mémoire (vive ou de stockage) nécessaire à son opération.

Vitesse Le but d'un cache est de réduire le temps nécessaire à l'affichage d'une page internet ; le logiciel présenté ne devra en aucun cas le ralentir.

Documentation Dans la mesure où ce cache est destiné à être déployé dans des pays en voie de développement, on veillera à ce que la qualité de la documentation soit exemplaire, afin de faciliter le transfert de connaissances à des techniciens locaux.

2.2 Objectifs secondaires

Stockage diversifié Dans le but de permettre une plus grande flexibilité sur les sites de déploiement, le cache pourrait, de plus, offrir la possibilité de profiter d'éventuels systèmes de stockage externes (bases de données déjà existante, systèmes de fichiers).

Auto-configuration Il serait intéressant de faciliter la configuration du cache quand il se trouve en présence d'un système de routage “intelligent”, de type OSPF. Ce protocole de routage permet en effet d'avoir une liste de noeuds voisins, dans la mesure où ce protocole la dresse déjà.

Génération de statistiques Il serait pratique pour la configuration de caches ultérieurs de pouvoir extraire des statistiques sur l'utilisation du cache : niveau de récursion moyen des requêtes, proportion de contenu servi depuis le cache, sites / pages / objets les plus consultés, type d'objets les plus demandés...

2.3 Facteurs de risque

2.3.1 Risque principal

Le risque principal dans la réalisation de ce logiciel est que les temps de réponse mesurés soient finalement, malgré une optimisation rigoureuse, plus longs que la requête originale

sans cache.

2.3.2 Risque secondaire

Un autre problème potentiel est que l'algorithme de parcours de graphe utilisé surcharge le réseau par un nombre d'appels récursifs trop grand. La détection et la correction des boucles est donc une priorité.

Chapitre 3

Cahier des charges détaillé

3.1 Livrables

3.1.1 Logiciel

La solution logicielle sera livrée sous la forme d'un CD contenant l'intégralité du code source de l'application, les binaires (le cas échéant), ainsi que tout autre fichier nécessaire au déploiement de ladite application.

3.1.2 Documentation

La documentation sera livrée en deux parties : une partie contenue dans le code source lui-même sous forme de commentaires, et une partie sous forme d'un fichier séparé, le manuel d'utilisateur.

Le manuel d'utilisateur doit permettre l'installation de l'application, son opération et sa maintenance sans aide externe. Les outils, librairies ou documents externes utilisés par l'application seront inclus, dans la mesure du possible et tant que leur licence de distribution le permet.

3.2 Fonctions souhaitées

Facilité de déploiement La documentation livrée doit permettre à un étudiant de premier cycle, non-programmeur mais familiarisé à la maintenance informatique, d'effectuer une installation complète sans autre aide que la documentation incluse.

Distribution Pour palier au problème des ressources disponibles, le cache sera distribué. Il devra néanmoins pouvoir fonctionner si aucun voisin n'est disponible.

Persistance Dans le cas d'une coupure de courant aléatoire, le cache doit, au redémarrage de la machine, se trouver dans le même état qu'avant celle-ci. Ceci implique que les données soient écrites de façon transactionnelle : soit dans une base de données, soit dans un système de fichier journalisé.

Configuration Le logiciel devra être configurable. On voudrait en particulier configurer : Les noeuds voisins, La quantité de mémoire vive allouée au cache, l'espace disque maximum à utiliser.

Filtrage des ressources On doit pouvoir filtrer les objets à mettre en cache, ainsi que le temps que chacun devra rester en cache.

Détection de boucles Le système devra être capable de détecter le plus vite possible les boucles réseaux qu'il crée (si il en crée), et régler le problème automatiquement, y compris en cas de mauvaise configuration par l'utilisateur.

3.3 Contraintes

Language La documentation et le code source devront être écrits en Anglais.

Vitesse d'exécution La vitesse d'exécution du logiciel, y-compris les requêtes sur les caches voisins et le retour de l'information de devront jamais dépasser le temps mesuré que met une requête, depuis un noeud donné, à obtenir l'information demandée depuis son serveur d'origine.

Utilisation des ressources Les ressources disponibles dans les pays émergeant sont bien moindres que celles dont nous avons l'habitude. L'application devra pouvoir fonctionner sur une machine de type Pentium (500Mhz), avec 128Mb de mémoire vive, et un espace disque de 256Mb dédié aux données. Le choix du système d'exploitation étant déterminant, on privilégiera une plate-forme *libre*, largement moins exigeante en ressources.

Liberté Le logiciel sera distribué sous licence GPLv3 selon les souhaits du mandant. Il est donc impératif que la licence en question soit respectée dans son intégralité.

Chapitre 4

Environnement de travail prévu

4.1 Matériel et outils

Aucun matériel spécifique n'est requis à l'exception du matériel habituel nécessaire à un programmeur (Ordinateur fonctionnel).

Les outils nécessaires sont : un outil d'édition de code source, un compilateur (ou un interpréteur le cas échéant), et une machine de déploiement. L'intégralité des outils sont disponibles en tant que logiciels libres pour n'importe quel système d'exploitation, le choix de ces derniers est donc peu significatif.

Le code source sera sauvegardé sur un système de gestion des sources, déjà existant et disponible (le présent document y est également sauvegardé).

4.2 Logiciel

4.2.1 Langage de programmation

Le choix se porte sur le Python, parce qu'il présente un certain nombre d'avantages intéressants :

Puissance : Python est un langage de script *orienté-objet* permettant la production de code concis et facile à comprendre. Il permet en outre d'enseigner la programmation de façon itérative, sans changer de langage : Commencer par de la programmation procédurale, puis retravailler le code pour le rendre *orienté-objet*, puis finalement introduire des notions de programmation "de haut niveau" (récursion, *lambda-fonctions*, introspection, programmation dynamique, programmation concurrente, algorithmes distribués, *design patterns* ...).

Syntaxe stricte : Python considère l'indentation comme un élément lexical. Le code python de n'importe quel programmeur au monde est donc indenté et formaté de la même façon, ce qui facilite grandement la compréhension d'un programme pour quelqu'un ayant déjà eu affaire à Python. Avoir appris Python permet donc de contribuer très rapidement à des projets *open-source*.

"Batteries included" ¹ : Python est fourni avec une librairie de fonctions et d'objets très vaste, et ne nécessite pas de faire appel à une ou des librairies externes pour une grande majorité de tâches. Ces concepteurs comparent ainsi Python à un appareil électronique dont les piles seraient vendues avec : nul besoin d'effectuer un achat séparé, tout est livré !

Vie active : Python est le langage de programmation de choix d'une vaste communauté. Un grand nombre de sociétés de développement logiciel contribuent activement à son évolution et à sa maintenance (Google, VMware, Apple, Microsoft, SUN, IBM). Beaucoup de projets *open-source* sont écrits en Python, il est donc facile de trouver des exemples, des sites communautaires d'échange de conseils, des *blogs* spécialisés...

¹ "Les piles sont livrées avec" - la version Anglaise est conservée car il s'agit d'une devise.

Multi-plateformes : Python est pré installé sur de très nombreux systèmes d’exploitation (MacOS, les Linux, Solaris, les BSD), et il est disponible en télé chargement pour de nombreux autres (Windows, AS400, etc.).

Extensible : Bien qu’étant un langage de scripting (donc comportant un coût d’*overhead*²), Python est aisément extensible et/ou optimisable avec C++, soit pour lui exposer une librairie non disponible dans la librairie standard, soit pour optimiser certains algorithmes ³

Documenté : Python est livré avec une documentation complète, et de nombreux tutoriaux de qualité exemplaire, faciles à comprendre et souvent pleins d’humour (ce qui facilite l’apprentissage).

4.2.2 Système d’exploitation

Le système d’exploitation de développement sera un système d’exploitation *libre*. Pour sa facilité de mise en place et sa grande flexibilité, le choix se porte sur Ubuntu GNU/Linux.

4.2.3 Base de données

Plusieurs solutions de stockage des données sont envisagées dans la planification, mais le logiciel devra dans tous les cas être capable de stocker ses données sur disque dur (afin de répondre à la partie “persistance” du cahier des charges, Section 2.1).

La solution la plus immédiate pour ce faire est d’utiliser le système de base de données *SQLite* (voir webographie, Section 7.1). Ce système offre plusieurs avantages particuliers, mais aussi certaines limitations décrites ci-après :

Avantages :

- Publique. SQLite est distribuée comme faisant partie du domaine publique. Quiconque est donc libre de faire ce que bon lui semble avec le code source. Bien que n’étant pas en tant que tel *libre*, SQLite offre certains des mêmes avantages qu’un logiciel *libre*, à savoir la liberté d’en étudier le code source, et la liberté de le redistribuer, avec ou sans modifications. Le site web de SQLite est cité en webographie (Section 7.1).
- Légère. Élaborée pour les systèmes embarqués, SQLite ne pèse que quelques ko sur disque.
- Rapide. SQLite permet de traiter efficacement une grande quantité de données (plusieurs Terra octets).
- Intégrée à Python. Les librairies d’accès à SQLite sont incluses dans la librairie de base de Python.
- ACID. Les versions les plus récentes de SQLite satisfont les exigences ACID⁴

Ses points faibles sont les suivants :

- Externe. Il est nécessaire d’installer SQLite avant d’utiliser le programme de cache. SQLite est en revanche préinstallée dans une majorité de systèmes d’exploitation *libres*, ce qui relativise ce désavantage.

4.3 Banc de test

Il existe plusieurs façons de simuler le déploiement d’un système distribué. En voici une énumération, ainsi qu’un bref aperçu des avantages et des inconvénients pour chaque cas.

²L’*overhead* est une faible surconsommation de ressources, sacrifiées au profit d’un autre avantage (ici, la facilité de rédaction et de lecture du code). Voir glossaire Section 8

³Il est possible de rendre ces algorithmes spécifiques à un type de processeur afin de tirer parti des possibilités *hardware*, comme par exemple d’instructions spécialisées

⁴ACID est l’acronyme Anglais de *Atomicity, Consistency, Isolation, Durability* (Voir Glossaire, Section 8)

4.3.1 Multiples hôtes physiques

Il s'agit de la façon la plus intuitive de tester le déploiement d'un réseau informatique : en créer un "in vivo". La technique ici est ni plus ni moins que de créer un véritable réseau avec de vraies machines. Évidemment, cette possibilité nécessite un nombre potentiellement grand de machine physiques disponibles. Ce n'est malheureusement pas le cas dans le cadre de ce travail.

4.3.2 Hôte unique, multiples instances

Une autre façon de modéliser un réseau est de faire fonctionner plusieurs copies du logiciel en parallèle sur la même machine, sur des ports différents. Un désavantage certain de la méthode est que le temps de latence sera toujours proche de zéro, or ce travail de diplôme par du postulat que le temps de latence est élevé dans l'environnement de déploiement ciblé. Cette stratégie conviendrait donc mieux pour tester un système de cache conventionnel, où la latence est un problème moins primordial.

4.3.3 Multiples hôtes virtuels

La virtualization permet de modéliser à moindre coût un réseau de potentiellement grand. Chaque machine virtuelle possède son propre système d'exploitation, ce qui permet de modéliser des réseaux hétérogènes. Les performances globales peuvent être bridées, et la répartition des ressources est garantie (par le fabricant) d'être équitable. Un ralentissement impacterait donc toutes les machines à la fois, ce qui permet de tester l'efficacité de l'algorithme, la complexité restant la même bien que le temps mesuré soit fluctuant. Ainsi, la vitesse d'exécution ralentit *de façon uniforme* pour toutes les machines virtuelles.

4.3.4 Solution hybride

Une solution hybride consisterai en l'application de plusieurs techniques sus-mentionnées à la fois : plusieurs hôtes physiques qui hébergeront chacun plusieurs machines virtuelles qui elles-mêmes feont fonctionner plusieurs copies du programme sur des ports différents, par exemple. Cette solution permet de créer des réseaux de test très complexes, ce qui semble disproportionné pour ce type de logiciel. De plus, bien que l'avantage de la taille soit certain, cette méthode cumule les désavantages des techniques utilisées.

4.3.5 Solution retenue pour ce travail

Pour ce travail en particulier, le choix se porte sur un système de test composé de multiples hôtes virtuels, comme décrit au point 4.3.3.

Les multiples machines virtuelles, bien qu'entraînant un ralentissement du système en général, permet au moins de garantir une dégradation homogène des performances entre les différentes machines virtuelles, ce qui permet de mesurer la performance d'un point de vue objectif. Les performances réelles seront donc proportionnellement meilleures que celles mesurées dans les tests.

Finalement, cette solution permet de simuler de façon plus réaliste les conditions de déploiement réelles, à savoir les performances matérielles définies dans la cahier des charges (section 3.3), en bridant les ressources allouées aux machines virtuelles pour qu'elles correspondent aux performances spécifiées.

Chapitre 5

Planification

La méthodologie de planification sera la planification “agile”¹. Dans ce sens, le travail sera découpé en 9 itérations de 2 semaines. Les objectifs seront découpés selon cette méthode en “*User stories*”², suivant les cas d’utilisation définis au préalable.

5.1 Plan de travail détaillé

5.1.1 Cas d’utilisation

On distingue ici 3 cas d’utilisations, avec 3 acteurs :

L’utilisateur Humain C’est un humain qui utilise l’application, éventuellement sans le savoir. Il se contente d’envoyer des requêtes sur internet, qui sont interceptées par notre cache.

L’administrateur L’administrateur est un humain qui installe l’application de cache, la configure, et l’utilise. En ce sens, on peut dire qu’un administrateur *étend* un utilisateur humain.

L’utilisateur cache voisin Cet utilisateur est un cache voisin qui envoie une demande de contenu, avec en header HTTP un niveau de récursion non null. Il est différent d’un utilisateur humain, car au cas où la requête n’est pas en cache, il faut réduire le niveau de récursion. Dans le cas d’un utilisateur humain il faut en plus envoyer une requête au serveur final.

5.1.2 Découpage du travail

Les user stories décrites englobent, comme le suggèrent les méthodes agiles, le temps consacré à l’écriture du code, mais aussi le temps passé au test et à la documentation. Une user story n’est considérée comme “finie” que lorsque le code, les tests et la documentation sont finis pour la fonctionnalité décrite. Dans ce projet, selon le cahier des charges, la documentation d’une user story comprend la rédaction de la partie du manuel utilisateur correspondante.

Dans les méthodes agiles de planification, on utilise une unité abstraite, les *story points* (ou “Points” en Français), qui sont généralement calculées pour chaque *user story* à partir d’une tâche-étalon en proportion de la difficulté qu’elle représente par rapport à cet étalon. Une équipe de développement choisit en général une tâche précédemment terminée

¹Les méthodologies agiles sont un ensemble de méthodes permettant la planification et la réalisation de logiciels de façon efficace. Se référer à la bibliographie en chapitre 7.2 pour de plus amples informations

²Les *user stories* sont les unités de découpage du travail de base utilisées dans les méthodes agiles. Voir glossaire au chapitre 8

dont toute l'équipe considère la difficulté comme faible.

Dans le cadre de ce travail en revanche, l'étalon utilisé pour déterminer les points sera le "jour idéal" de travail, unité généralement utilisée quand une équipe travaille ensemble pour la première fois, et qui donc n'a pas déjà effectué ensemble de tâche à utiliser comme étalon. Un jour idéal de travail représente une journée "parfaite", sans séances, et durant laquelle le programmeur peut travailler à cette unique tâche sans être interrompu par un événement extérieur. Naturellement, une telle journée est incroyablement rare dans une entreprise normale, mais l'étalon est néanmoins valide, puisqu'il est par nature commun aux membres de l'équipe. Ici, le "jour idéal" doit être compris comme "soirée idéale", puisque le travail sera effectué à temps partiel.

- Une soirée dure 4 heures.
- Il y a 4 soirées par semaines.
- Il y a 4 semaines par mois, donc 16 soirées idéales
- Le travail s'étend sur 4 mois, il y a donc 64 soirées idéales.

Les heures de travail n'étant jamais idéales, le projet s'étendra donc logiquement sur une durée légèrement supérieure. Cette approche permet néanmoins d'estimer au fur et à mesure que le projet avance de la quantité de travail effectuée (en proportion), ainsi que la vitesse moyenne de progression. Cette vitesse est appelée *vélocité* dans le "jargon agile", et sera calculée à la fin de l'itération, afin de modifier, au besoin, le planning des itérations suivantes. En entreprise, cela permet en outre au client de changer les priorités des tâches, et d'avoir un état d'avancement général du projet (en pourcentage).

5.1.3 User stories

Le travail de programmation sera divisé en *user stories* comme suit :

1	En tant qu'utilisateur, je veux un cache Web local	<i>Points : 5</i>
Cache web local sur port 8080, renvoie des requêtes sur la cible si l'objet n'est pas en cache		
<i>Priorité business : Haut</i>		<i>Effort : Haut</i>
2	En tant qu'utilisateur, je veux que le cache soit persistant	<i>Points : 7</i>
– Stocque les objets sur disque dans SQLite – le cache doit être nettoyé quand la limite de place est atteinte ou quand l'objet est présent mais trop vieux.		
<i>Priorité business : Haut</i>		<i>Effort : Bas</i>
3	En tant que serveur de cache, je veux utiliser des marqueurs de récursion pour le dialogue avec mes pairs	<i>Points : 6</i>
Si la requête originelle contient un marqueur de récursion et que les données sont en cache, répondre. Sinon, la transmettre aux pairs en décrémentant le marqueur de récursion de 1.		
<i>Priorité business : Haut</i>		<i>Effort : Bas</i>

4	En tant qu'Administrateur, je veux configurer mon cache à l'aide d'un fichier de configuration	<i>Points : 10</i>
Le fichier doit contenir au moins les informations suivantes : – Liste des caches pairs – Port d'écoute – Niveau de récursion désiré – Règles de fraîcheur - extensions fichiers, temps maximum de fraîcheur...		
<i>Priorité business : Haut</i>		<i>Effort : Haut</i>
5	En tant qu'administrateur, je veux conserver les données dans une base externe (MySQL)	<i>Points : 7</i>
Vérifier que les connecteurs SQLite peuvent facilement être échangés pour des connecteurs MySQL.		
<i>Priorité business : Bas</i>		<i>Effort : Bas</i>
6	En tant qu'Administrateur, je veux conserver les données du cache dans le système de fichiers	<i>Points : 9</i>
– Le système de fichiers doit être journalisé et transactionnel (ext3 par exemple) – Distribution des objets dans un arbre binaire de profondeur X (paramétrable) avec les deux premiers symboles Hexadécimaux des données ou d'un hash des données comme nom de dossier		
<i>Priorité business : Bas</i>		<i>Effort : Haut</i>
7	En tant qu'Administrateur, je veux que le cache utilise les données de mon réseau OSPF pour trouver ses pairs	<i>Points : 13</i>
Tâche très compliquée techniquement ! – Nécessite la mise en place d'un réseau OSPF de test – Extraire les données du routeur local à la machine au démarrage du cache – Prévoir une entrée dans le fichier de configuration – Systèmes à base UNIX uniquement ! OSPF pour Windows pas disponible		
<i>Priorité business : Bas</i>		<i>Effort : Haut</i>
8	En tant qu'Administrateur, je veux des statistiques d'utilisation du cache	<i>Points : 7</i>
Les statistiques suivantes seraient intéressantes : – Niveau de recursion moyen des requêtes – Proportion de contenu servi depuis le cache (<i>hit-rate</i>) – Objets les plus consultés – Sites les plus consultés – Type d'objets les plus demandés		
<i>Priorité business : Bas</i>		<i>Effort : Haut</i>

5.1.4 Notes sur les user stories

Les *user stories* décrites ci-dessus sont estimées à un total de 64 soirées idéales. Il est courant de planifier plus de points que disponible, pour la raison simple que ces points ne sont qu'une *estimation* de la quantité de travail nécessaire. Il est fort possible qu'une user story soit finalement plus facile à réaliser que prévu.

De plus, les user stories incluent des objectifs secondaires, dont la priorité est moindre. Les méthodes agiles privilégient la livraison d'un logiciel à temps, même si toutes les fonctionnalités ne sont pas terminées. Ainsi, il est possible pour Ingénieurs du Monde d'utiliser le

logiciel dans son contexte de déploiement réel, même si la *user story* numéro 5 (Sauvegarder les données dans une base MySQL) n'est pas utilisable. En revanche, il est impossible pour eux d'utiliser le logiciel si la *user story* numéro 1 (Un cache web local) n'est pas finie !

Les user stories identifiées comme non-cruciales, qui répondent aux objectifs secondaires, seront donc terminées *si le temps le permet*. Il est donc normal d'avoir plus de Points planifiés que de soirées réelles (pas idéales), mais il faut garantir que les objectifs principaux seront réalisés. Cette différence de priorité est exprimée dans les "cartes" des user stories sous la forme de *Priorité Business*. Les priorités hautes sont nécessaires à la réussite du projet (et correspondent aux objectifs principaux), les autres correspondent aux objectifs secondaires et peuvent éventuellement ne pas être réalisées.

Bien entendu, l'idéal est qu'elles soient toutes réalisées à temps !

5.1.5 Découpage temporel

Le travail est découpé en plusieurs phases temporelles :

Pré-Etude (Phase d'analyse) C'est la phase de travail qui vise à produire le présent document. Elle s'étend du 6 Avril au 5 Juin.

Phase de réalisation La phase de fabrication du logiciel. Cette phase sera planifiée et mise en place en utilisant les méthodes agiles. Ainsi, on découpera la période de réalisation en *itérations*³.

Phase de mise en production Dernière partie de la planification, elle consiste à finaliser le projet, régler d'éventuels derniers bugs, s'occuper de la distribution du produit, et préparer avec le mandant la mise en production. Cette phase se situe après l'échéance scolaire, mais est néanmoins mentionnée ici dans la mesure où elle fait partie du projet.

5.1.6 Découpage de la phase de réalisation en itérations

La phase de réalisation du projet, qui s'étend du 29 Juin au 30 Octobre, sera découpée en 4 itérations de 1 mois. Fidèle aux méthodes agiles, un certain nombre de user-stories seront attribuées à chaque iteration en fonction de l'estimation du temps de réalisation, et en commençant par les tâches à priori les plus "risquées", c'est à dire les plus critiques ou difficiles à réaliser. L'objectif de cette répartition est de détecter les problèmes le plus vite possible, afin d'avoir un maximum de marge de manoeuvre pour les régler, les contourner ou les éviter.

Les itérations ont une *vélocité* de 16 Points ("soirées idéales").

5.1.7 Assignation des user stories aux différentes itérations

Comme décrit précédemment, les user stories les plus risquées devront être traitées en priorité, afin de se garantir une plus grande marge de manoeuvre face aux problèmes éventuels.

Itération 1 Le thème de cette première itération est "Créer un cache web persistant" :

- User story 1 (*En tant qu'utilisateur je veux un cache Web local*)
- User story 2 (*En tant qu'utilisateur, je veux que le cache soit persistant*)

Cette itération comptabilise 12 points idéaux. C'est moins que la *vélocité*, mais elle devra en outre prendre en compte le temps de mise en place du projet, de l'environnement de développement et de celui de test.

Itération 2 Cette seconde itération a pour but de rendre le cache "intelligent et configurable" :

- User story 3 (*En tant que serveur de cache, je veux utiliser des marqueurs de récursion pour le dialogue avec mes pairs*)

³Une itération est une période de temps à la fin duquel on dispose d'un produit livrable, bien que n'ayant pas toutes les fonctionnalités requises. Voir glossaire (Section 8)

-
- User story 4 (*En tant qu'Administrateur, je veux configurer mon cache à l'aide d'un fichier de configuration*)

La difficulté réside ici dans l'implémentation de la logique du cache, c'est le coeur du problème. Cependant, l'environnement est déjà en place et le programmeur familiarisé avec le code existant.

Itération 3 Ici, l'itération s'occupe d'un objectif secondaire, “Diversifier la sauvegarde des données” :

- User story 5 (*En tant qu'administrateur, je veux conserver les données dans une base externe MySQL*)
- User story 6 (*En tant qu'Administrateur, je veux conserver les données du cache dans le système de fichiers*)

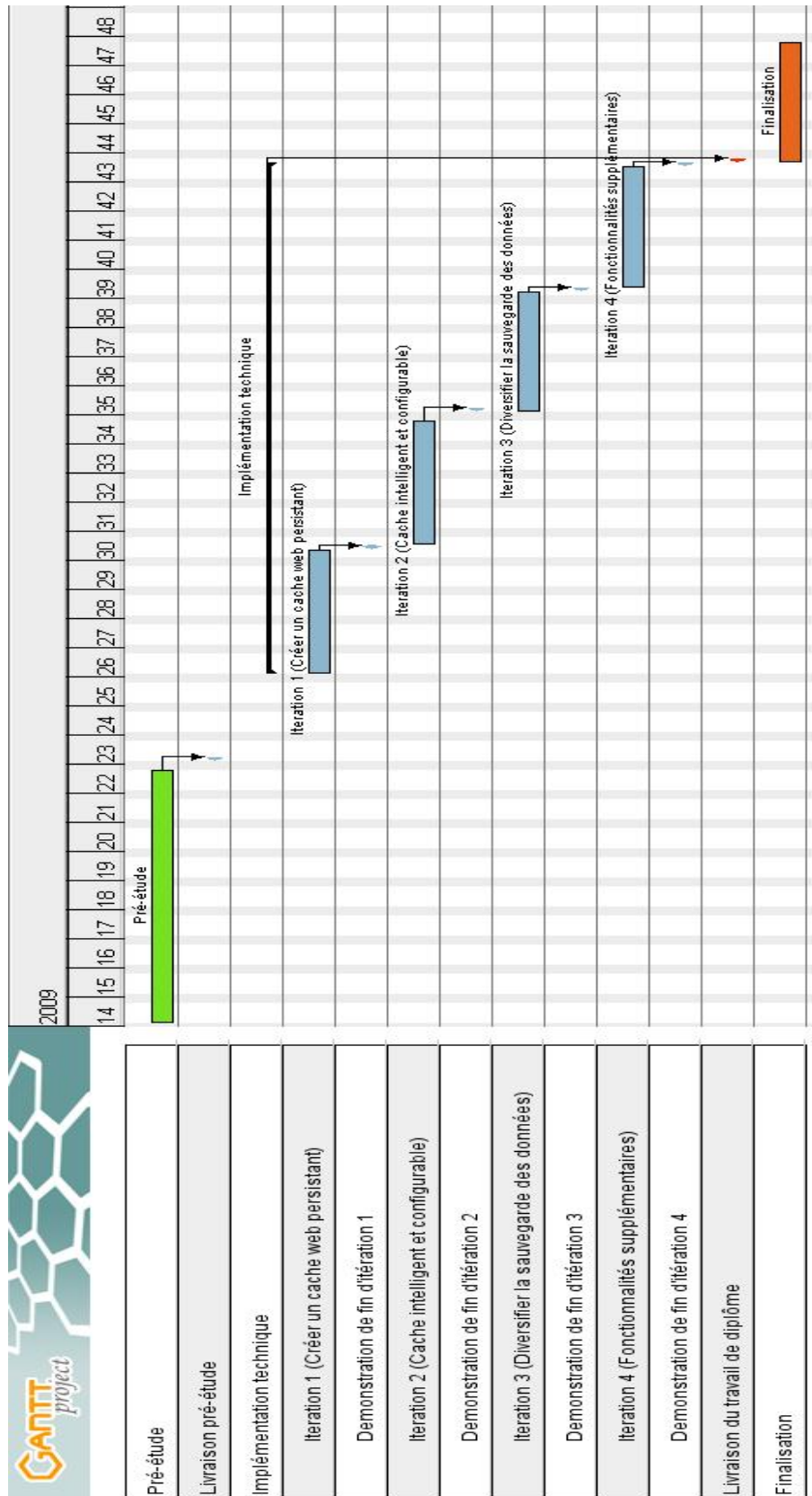
Première itération consacrée aux objectifs secondaires, facultatifs. On prendra garde aux coûts en performance des systèmes de sauvegarde.

Itération 4 D'autres objectifs secondaires, axés sur les “fonctionnalités supplémentaires” :

- User story 7 (*En tant qu'Administrateur, je veux que le cache utilise les données de mon réseau OSPF pour trouver ses pairs*)
- User story 8 (*En tant qu'Administrateur, je veux des statistiques d'utilisation de mon cache*)

La difficulté sera principalement de mettre en place un réseau de test OSPF.

5.2 Échéancier



Chapitre 6

Conclusion

6.1 Remerciements

Tout d’abord, merci à mon conseiller le Dr. Zysman pour le temps qu’il a investi et la patience dont il a su faire preuve à mon égard.

J’aimerais aussi remercier le Dr. Ullmann et toute l’équipe d’Ingénieurs du Monde pour l’intérêt et le sérieux qu’ils ont accordé à ce projet. Il est intimidant de défendre un projet d’études devant un tel nombre de Docteurs *ès* Sciences dans des domaines divers, mais tous ont sû être rassurants et encourageants lors de nos échanges.

De même, je tiens à remercier également mon collègue et camarade Oliver Amblet pour son aide lors de l’estimation des tâches et du découpage du planning. Chef de projet dans nos bureaux de Lausanne, Oliver a su me faire profiter de son expérience et de ses conseils pour la planification, et m’a ainsi permis d’éviter plusieurs pièges courants, notamment de sur-spécifier en définissant des user-stories trop ciblées, et de faire des itérations trop petites, piège facile quand on a l’habitude de travailler à plein temps sur un projet et non pas à temps partiel comme c’est le cas ici.

Finalement, un grand merci au Dr. Daniel Siemaszko, de l’Institut Royal de Technologies de Stockholm. Ami de longue date, Daniel a eu l’amabilité de jeter un oeil critique, fruit de sa longue expérience en écriture de rapports scientifiques, sur cette pré-étude en général et sur mon utilisation de LaTeX en particulier.

6.2 Pourquoi ce projet ?

Ce projet de recherche permet d’explorer de nombreux aspects intéressants de l’informatique, et non le seul aspect technique auquel un programmeur est le plus souvent confronté.

Dans notre bulle occidentale largement interconnectée où l’information circule rapidement en utilisant des technologies de pointes, très chère, le programmeur moderne perd souvent de vue la nécessité d’adapter les algorithmes et les procédés à des infrastructures dont les limitations matérielles sont plus contraignantes, comme celles rencontrées les plus souvent dans les pays en voie de développement. Cette barrière économique et psychologique est un énorme frein au développement de réseaux informatiques dans ces régions, qui ne disposent pas des moyens financiers nécessaires à la mise à niveau occidentale de leurs infrastructures, et pour lesquelles les solutions “astucieuses” en occident ne sont pas adaptées.

Un système de cache distribué adapté à cette configuration n’existe pas, à ma connaissance, et le fait d’en inventer un de toutes pièces représente un défi de taille qu’il me plaît

de relever.

Employé dans une société multinationale dont les produits s'adressent à d'autres multinationales prêtes à dépenser d'importantes sommes d'argent pour acquérir des technologies très "pointues", ce projet est pour moi une opportunité de penser différemment, et de laisser une part de culot et de créativité à ma conception logicielle. Avec le concours d'Ingénieurs du Monde, ce logiciel pourra je l'espère se voir déployer autre part que sur un réseau de test, et surtout répondre au besoin de personnes qui un jour, écriront à leur tour un rapport de pré-étude en vue de l'obtention de leur diplôme d'ingénieurs.

Ce travail, purement logiciel, n'est qu'une petite partie des technologies nécessaires à la mise en place d'un réseau internet. J'espère pouvoir continuer à contribuer, dans la mesure de mes capacités, à la mise en place des autres "briques" nécessaires au déploiement d'un réseau WiFi à grande échelle : conception d'antennes plus efficaces, dessin des mâts, analyse de terrain pour déterminer les meilleurs emplacements, étude des besoins *hardware* de chaque antenne, analyse des procédés de refroidissement des machines, des besoins en alimentation électrique...Quantité de problèmes restent à résoudre !

Le défi est passionnant.

Chapitre 7

Exploitation de sources bibliographiques

7.1 Webographie

- Ingénieurs du Monde, “Ingénieurs du Monde”
<http://www.ingenieursdumonde.org/> (Page consultée le 1er Juin 2009)
- Weijand R&D Consultancy BV, “10 Euro dish with biquad feeder”
<http://www.weijand.nl/wifi/> (page consultée le 1er Juin 2009).
- The Python foundation, “Python documentation”
<http://www.python.org/doc/> (page consultée fréquemment... mais pour les besoins de la référence, en date du 1er Juin 2009).
- The squid-cache.org team, “Squid-Cache.org”
<http://www.squid-cache.org> (page consultée le 1er Juin 2009)
- The Dijjer team, “Dijjer”
<http://code.google.com/p/dijjer/> (page consultée le 1er Juin 2009)
- Tim Love (And the Cambridge University Engineering Dept.), “Text Processing using LaTeX”
<http://www.eng.cam.ac.uk/help/tpl/textprocessing/> (page consultée le 1er Juin 2009)
- BitTorrent Inc., “What Is Bittorrent”
<http://www.bittorrent.com/btusers/what-is-bittorrent> (page consultée le 1er Juin 2009)
- The SQLite.org team, “SQLite”
<http://www.sqlite.org> (page consultée le 1er Juin 2009)

7.2 Bibliographie

- Mike Cohn, *Agile estimating and planning*, Pearson Education Inc., 2006.
- Mark Summerfield, *Programming in Python 3 : A Complete Introduction to the Python Language*, Addison-Wesley Professional, Décembre 2008.
- Guido van Rossum, and Fred L. Drake, Jr., *An Introduction to Python (version 2.5)*, Network Theory Ltd, Novembre 2006.
- John Goerzen, *Foundations of Python Network Programming*, Apress Inc, Août 2004.
- Tarek Ziadé, *Expert Python Programming*, Packt Publishing, Septembre 2008.

Chapitre 8

Glossaire

ACID Acronyme Anglais de *Atomicity, Consistency, Isolation, Durability* (Atomicité, Consistance, Isolation, Durabilité). Propriétés garantissant qu’une transaction de base de données soit effectuée de manière sûres (sans pertes de données, sans collisions).

Bande passante Quantité de données transmises en un temps donné. Typiquement exprimée en kilo bits (kb) par seconde (kb/s), ou en mégabits (mb) par seconde (mb/s).

Bittorrent Protocole spécialisé dans l’échange de fichiers entre un grand nombre de personnes. Bittorrent permet à ses utilisateurs de publier un descriptif des fichiers qu’ils souhaitent échanger sur un serveur dédié (un *tracker* Bittorrent). Chaque utilisateur se connecte au *tracker*, choisit les fichiers qu’il souhaite obtenir, et publie la liste des fichiers qu’il souhaite distribuer.

Lors du téléchargement à proprement parler, les fichiers sont découpés en petits morceaux de quelques ko (kilo octets). Chaque utilisateur se connecte alors aux utilisateurs qui ont annoncé posséder tel ou tel morceau de fichier, et le télécharge directement depuis la machine de l’autre. Avantage : quand je télécharge un fichier, je télécharge en fait des morceaux de fichier depuis plusieurs autres utilisateurs à la fois. Bittorrent vérifie leur intégrité et recompose le fichier une fois tous ses morceaux récupérés.

Cache Un *Cache* en informatique est un lieu (en mémoire) de stockage temporaire du duplicata d’une information considérée trop chère en ressources à récupérer à nouveau ou à recalculer. Il existe quantité d’endroits où des données sont *cachées* dans un ordinateur moderne (processeur, périphériques réseau, cartes son, disques durs...). Dans le cadre de ce travail, *cache* a toujours le sens de *cache réseau*. Il s’agit d’un système permettant le stockage temporaire de pages internet et de contenu assimilé, telles que images, feuilles de style, scripts, objets multimédias, afin d’accélérer l’affichage de l’information la seconde fois qu’un utilisateur demandera le même objet.

Ainsi, si deux personnes, sur deux ordinateurs différents partageant le même *cache* accèdent à la même page internet de façon séquentielle, le premier utilisateur recevra la page depuis le site originel (vraisemblablement à distance), alors que le second, demandant la même page, recevra le contenu depuis le *cache*. Le *cache* aura intercepté la demande du premier utilisateur et stocké le résultat (*mis en cache*). Le second utilisateur aura donc vu la page s’afficher beaucoup plus vite que le premier, étant donné que les données étaient locales au réseau dont l’utilisateur fait parti.

Gateway Mot Anglais signifiant “Portail”. Un ordinateur *gateway* est un ordinateur appartenant à deux réseaux différents, qui utilisent potentiellement deux protocoles différents, et qui permet aux membres des deux réseaux de communiquer entre eux. C’est la porte par laquelle les deux réseaux communiquent.

Niveau de récursion Le niveau de récursion est ici utilisé au sens de la théorie des graphes, c’est à dire, en partant d’un noeud N appartenant au graphe G, le niveau de récursion est la profondeur d’exploration de l’arbre recouvrant de G ayant N pour racine. Plus généralement, la récursion est le fait pour une fonction informatique de s’appeler elle-même. Dans notre cas, la fonction en question est la fonction de transmission de l’information. Ainsi, une fois appliquée à un noeud “père” le noeud va lui-même appeler

cette fonction sur ses “fils”, qui eux-même l’appliqueront à leurs “petits-fils”, et ainsi de suite.

Temps de latence Aussi appelé *lag*, le temps de latence désigne ici le temps écoulé entre l’émission d’une requête et la réception de la réponse correspondante par le serveur interrogé. Plus généralement, le temps de latence désigne le temps écoulé entre un événement et le début de la réaction à celui-ci.

OSPF Acronyme Anglais de *Open Shortest Path First*. Protocole de routage réseau où chaque noeud (routeur) établit une carte d’adjacence avec ses voisins, et la leur transmet. Il reçoit à son tour la carte d’adjacence de ses voisins, ce qui lui permet d’avoir une vision globale du réseau auquel il appartient. Le protocole lui permet ensuite d’envoyer un message à un noeud quelconque de façon efficace, en appliquant l’algorithme de Dijkstra (pour trouver le chemin le plus court entre deux points dans un graphe non orienté).

Overhead De l’Anglais, littéralement “Au dessus de la tête”. Faible surconsommation de ressources au profit d’un avantage quelconque. Un *overhead* n’est donc pas considéré comme une perte, mais comme un sacrifice mesuré. On parle par exemple de l’*overhead* d’un protocole réseau qui sacrifierait un peu de temps de calcul afin de garantir l’intégrité des données. Encore, un langage de programmation (ici, Python) peut sacrifier un peu de performances au profit de la lisibilité du code source ou de la flexibilité de la syntaxe.

User stories De l’Anglais, “histoires utilisateur”. Unité de base pour la découpe et la répartition du travail dans les équipes utilisant les méthodologies agiles. Une *user story* est composée des éléments suivants :

- Un titre, reprenant un acteur des cas d’utilisation : *As an actor, I want...*
- Un numéro d’identification. Lors de gros travaux, il n’est pas rare d’avoir plusieurs centaines de *user stories*. Les numéroter permet de les classer et les référencer de façon efficace.
- L’estimation de la difficulté de la tâche
- L’estimation de la priorité de la tâche

Les user stories sont découpées plus finement en tâches au début de l’itération où le travail qu’elles représentent est commencé. Typiquement, les tâches sont ensuite réparties entre les programmeurs à leur bon vouloir, chacun s’attribuant la tâche qui lui “plaît” le plus ou celle pour laquelle il se considère le plus compétent. À la fin d’une tâche, le programmeur choisit sa tâche suivante, jusqu’à ce qu’il ne reste plus de tâches pour l’itération courante. Cet artifice permet de découper agréablement la vision que chaque membre de l’équipe se fait du projet : le chef de projet s’occupe d’assigner les user stories aux différentes équipes, sans se soucier des détails techniques. Les équipes se concentrent sur les tâches en gérant elles-mêmes la façon dont elles sont réparties.